**IMPORTANT PYTORCH DISCUSSIONS AND ANSWERS TO ALL KIND OF QUESTIONS**



## **[Matthew Spire [May 31st at 8:29 AM]] So, in the video, he says multiplying the features and the weights, but aren't you actually taking the dot product? So, you have a tensor of features [m x n] and a tensor of weights [m x n], but the tensor of weights is transposed before the math is done. Am I missing something? How does weights get transposed?**

Hey actually we write y = wT\*x + b normally where we take transpose for weights and other than that you can also see that to keep the dimensions equall throughout we have to do this

if it helps, the `\*` operation is an element-wise multiplication operator. If `features` & `weights` are both `1 x m` row vectors, then `features \* weights` is a `1 x m` row vector is as well

**so, \* operator will make the transpose of weights automatically?**

no. usually, you'd normally use `torch.mm` or something like that for matrix multiplication instead. Rather, it's just that `torch.sum (A \* B)` amounts to being the dot product of A and B. They used `\*` here only because both `features` and `weights` are row vectors (not true in later lessons or notebooks).

## **why my Workspace is in Chinese language? how to change to English?**

Only first notebook is in Chinese. check the second one. it is in english. Change the settings into english. (edited

Go to the top most botton to the right, click on it, go to setting, go to general you will see language and region then select English

click on your name in upper left corner then click third option it will take you to preferences. then second option for language.

If you have a notebook displaying Chinese characters and you don't want this, select the notebook with the same name, but without a '-cn' prefix and you are good to go.

@channel

Hello all! Those who are facing issues with Chinese notebooks in workspaces

1. Click on \*jupyter\* icon

2. Then you will see a lot more notebooks in that folder

3. Note that, the notebooks with \*-cn\* at last refers to Chinese.

4. Who want to go with English select notebooks without \*-cn\* at the end of notebook name.

## **Question: Does PyTorch use Numpy under the hood for performing operations? It seems a possible scenario, when you can switch between both PyTorch and Numpy very easily**

No pytorch does not use numpy under the hood. For tensor optimisation,they need to be put on a GPU. Numpy is CPU optimised.

Yes and you can convert from tensor to numpy or numpy to tensor

Yeah, but you can optimise Numpy for GPU operations by using a small wrapper called CUDA-MAT, this way Numpy operations can run on cuda cores

<http://blog.christianperone.com/2018/03/pytorch-internal-architecture-tour/>

## **What do you mean by torch.manual\_seed(7) ???**

seed is generally used as input to random number generators.

You just need to call torch.manual\_seed(seed), and it will set the seed of the random number generator to a fixed value, so that when you call for example torch.rand(2), the results will be reproducible.

It is generally a good practice to set seeds, so that you can reproduce, or even compare between results.

**can u explain the word seed by a bit more what does it actually mean**

**"A seed is an initial value that is fed into a pseudo random number generator to start or kick off the process of random number generation. " source – quora**

Randomness in computers is inherently not random. Computers use a number (seed) to start the generation of random numbers. By setting this seed to a constant you ensure that the process is constraint to that specific instance. While training neural networks a myriad of factors can cause you to get results that vary from each training. Therefore, to keep the results the same and reproducible, it is generally good practice to set a random seed.

## **So we've seen that PyTorch uses some C Backend defined in THTensorMath.c to perform matrix multiplications right?**

My doubt is in case we have a large multidimensional data set to be trained (like a 1080p videos or something), I think that certainly the values obtained on multiplying would exceed any of C language data types size, so how does PyTorch perform operations when dealing with such data to give accurate results ?

<https://apaszke.github.io/torch-internals.html>

Videos are made of frames/pictures. Instead of loading all the frames of video at once. You might want to load 1 or few frames and do your operations. A basic example, if you are doing object detection on a video you can do inference on a single frame and continue to repeat the process.

The challenge in keeping the images large is that the deep learning model size explodes. If you are operating on an image that is 2,000 by 2,000 pixels for example, each layer can have millions of nodes.

PCIe Bottleneck for Large Models

When the model size becomes very large because of the size of input data, it is not possible to keep the entire model in the GPU. So, you have to keep the model and data in the system memory connected to the CPU and move over pieces at a time to the GPU. The PCI-Express (PCIe) connection between the CPU and GPU, however, becomes a bottleneck for this communication.

, just like every other machine learning technique, we need to do the preprocessing when we are doing inference

<https://medium.com/@sumitg_16893/deep-learning-with-high-resolution-images-large-models-44bfd90482a8>

The above article talks about hardware that lets you use bigger image sizes for higher accuracy.

Its an interesting read.

## **Data augmentation ?**

<https://pytorch.org/docs/stable/torchvision/transforms.html>

Data augmentation changes the the original image so that the model get the general idea about the image. It crops,rotates,resizes the image. It can be used to prevent overfitting also

it just creates randomness in our training data so that it does not overfit our training data. Hope this helps

Not just randomness, it simply creates many different perspectives on the same images. Deep Learning is more like a Brute Force approach to AI learning. Our Model learns by analysing pictures from as many perspectives and as many environments as possible. Data augmentation simply allows us to widen the range of perspectives we have, allowing the model to better adjust to these possibility's

to answer your question of how data augmentation helps with overtraining: data augmentation is the process of using the data you currently have and modifying it in a realistic but randomised way, to increase the variety of data seen during training. As an example for images, slightly rotating, zooming, and/or translating the image will result in the same content, but with a different framing. This is representative of the real-world scenario, so will improve the training.

@Dustin Gogoll I believe there are approaches in AI that mimics human behavior and continue to learn (example reinforcement learning) If the model is kept fixed your argument may be valid though. It's up to us engineers to decide retrain model on oncoming new data when deemed suitable.

And again I view ML approaches as generalization of whatever available data so that it could cope with somewhat different yet similar to train data. So it's not entirely bruteforce per se.

That's my viewpoint. Any different viewpoints are welcome

You are correct, thats why I was talking about Deep Learning itself. For example, there are techniques like semi supervised learning, which is less of an Brute Force approach, and there is, what you already said, reinforcement learning and many other techniques. But if we simply talk about Deep Learning in it's core it's more like an Brute Force approach, and I would definitely stick to that

I agree with your opinion. Deep learning more of brute force approach. In deep learning you give the model loads of data and it learns by backpropagation. There is no clear explanation how it is learning. It only backpropagate through error and adjust the weight so loss can be minimised. Deep learning is the student who almost give the right answer but it doesn't know how to get the answers.

## **Does anyone know how to set up a CUDA and PyTorch environment on Google Cloud AI Platform? I'm confused on whether the framework for my new notebook instance should be PyTorch or CUDA (it only allows me to choose one). Alternatively, does anyone know how to setup Jupyter Notebook on Google Cloud VM instance?**

Use PyTorch as framework, Nvidia drivers are pre installed I Suppose

Follow this one <https://cloud.google.com/deep-learning-vm/docs/pytorch_start_instance>

Easiest way if you want to use GCP VMs is Cloud Datalab: Open Cloudshell -> 'datalab create dataengvm --zone us-central1-a' -> webpreview port 8081

Alternatively you can use Google Colab from https://colab.research.google.com/

Or use the Udacity GPU Notebooks from the course.

## **can anyone help with the arguments beta1 and beta2 which are passed in torch.optim.Adam ? What's the use of those parameters ?**

Adam optimizer uses learning rate decay (slowly decreasing the learning rate) to try to get to the global minima and the parameters beta1 and beta2 are the different exponential decay rates.

<https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/>

<https://www.youtube.com/watch?v=1waHlpKiNyY&list=PLkDaE6sCZn6Hn0vK8co82zjQtt3T2Nkqc>

have a look at this playlist from video 17-22 you'll have better idea

the parameters beta1 and beta2 control the decay rates of the moving averages. First adam

They are tunable decay parameters for the Adam Optimizer, for in-depth please refer the Adam Algorithm https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

## **can anyone suggest a good step by step material for setting up Cuda for PyTorch in a windows 10 machine.**

. Check if your graphics card supports cuda, if yes download the proper version from Nvidia's website for that specific card.

This is a pretty good baseline article for it:

https://towardsdatascience.com/installing-tensorflow-with-cuda-cudnn-and-gpu-support-on-windows-10-60693e46e781

Although the official doc is here:

<https://docs.nvidia.com/cuda/cuda-installation-guide-microsoft-windows/index.html>

Install these :

1. Install CUDA toolkit : https://developer.nvidia.com/cuda-downloads?target\_os=Windows&target\_arch=x86\_64

2. Install cuDNN https://developer.nvidia.com/cudnn

3. Install conda : https://www.anaconda.com/distribution/

4. Then install pytorch : <https://pytorch.org/>

My doubt is, by installing pytorch with anaconda, do we need to separately install cuda and cudnn before installing pytorch or it is automatically installed during the pytorch installation process.

you need to install cuda and cudnn beforehand to install pytorch gpu version using conda

I tried import torch in Colab is doesn't really work. Now just trying to install OpenCL first.

you may have to install it first before using it.

You can follow the instructions here : <https://jovianlin.io/pytorch-with-gpu-in-google-colab/>

## **What is the function of the activation function ?**

Neural networks are here just because of the fact that they can learn non-linear functions, an activation function does just that by adding non-linearity to the network

Also without activation function our Neural network would not be able to learn and model other complicated kinds of data such as images, videos , audio , speech etc.

f you don't add an activation function, a neural net will be only learning linear functions and estimating values for the same, which by the way; any regressor model can do

They will convert input into a range of values e.g relu will convert -ve numbers into 0 and will do nothing to +ve numbers. (

Activation function allows the model to learn complex features. So, as you can see the term (WX + B) is a linear function. So, if we don't apply any non linear activation functions then the output of every layer would be linear. So our model won't learn any complex features. The common types of activation functions are sigmoid, tanh, relu and relu leaky. I think the most common used activation function is the relu activation for the hidden layer.

An activation function can either be linear or non linear. If a dataset is linearly separable then we would not use non-linear activation function ( separable by using a straight line)

If a dataset is not linearly separable ( consider two types of seeds mixed up), then you will have to apply non linear activation function ( here you can assume a curvy plane that will help you to figure out which seed belong to which type).

Note: Having some 10 hidden layer with linear activation function will be equivalent to one single layer perceptron, as adding all those 10 layers would give you just another linear function.

To practically see what happens, just create a perceptron using numpy and don't apply any activation function. And another with non linear activation function. See what would happen if you train and test it on a planar data.

Kindly let me know if any of the above points are incorrect/misinterpreted.

yea that's what machine learning is. Learning the function that can map inputs to output. Linear regression is used when the relationship is linear. When the relationship is much more complex you use neural nets. Activation functions introduce non linearity in the function. In the simplest sense neural nets with many layers and activation functions are just really good function approximators when there is a complex relationship.

## **Instead of use of `weights.reshape(), weights.resize\_()` you can clearly use `torch.t` that transpose the matrix from row to column or from column to row**

## **[Ayush Jain ] What does \*top\_class.shape mean? Why is there an asterisk? equals = top\_class == labels.view(\*top\_class.shape)**

the \* returns the elements of the list as single elements separately

abels.view() is like reshape in numpy,

top\_class.shape is the shape of the tensor

when \*top\_class.shape is placed inside label.view it means it doing reshaping tensor label to be the same like top\_class tensor

shape returns a tuple. by adding the asterisk, the tuple gets "unpacked" and you receive two separate arguments

https://link.medium.com/0OYS4cJzgX

## **If we are not calculating the gradient and optimizing the model parameters what's the use of initialising an optimizer in the code for validation of the test and validation data..?**

The gradients are calculated by the loss.backward() and weights are updated using optimizer.step(). Hence, there is no need in the test and validation because they are used to see the performance of the model.

You haven't used optimizer.step() in validation step. It is only used in training step.

In this code, there are two parts:

the for loop - it does the training, and only during the for the optimizer is used.

the else statement - it uses the validation dataset to the validation step.

In validation we only calculate loss we don't use optimiser.step().

We should not caculate gradients during validation or testing phase because if you do that then you will fails to optimize your model because the model learn data from validation as well. So we just pass test data through model and calculate the loss. Based on the loss we update the model with other parameters.

During training we need optimizer and for test and validatiin we need loss function.

We are only using the optimizer for training the model as optimization helps us to get the best value of our model parameters. Check the code of your validation case, we haven't used any optimizer there.

## **How we can approximate the number of convolution layer, filters per convolution, weights per filters? can any one provide a useful resource on what each layer may visualize what per kernel is visualize in general and not just for example provided in this lesson ?**

[**https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks**](https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks)

[**http://www.isikdogan.com/blog/how-to-design-a-convolutional-neural-network.html**](http://www.isikdogan.com/blog/how-to-design-a-convolutional-neural-network.html)

**https://www.youtube.com/playlist?list=PLkDaE6sCZn6Gl29AoE31iwdVwSG-KnDzF**

**@Shymaa Andrew NG's course on CNNs**

[**http://cs231n.github.io/convolutional-networks/#conv**](http://cs231n.github.io/convolutional-networks/#conv)

[**https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53**](https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53)

[**https://youtu.be/0r9o8hprDXQ**](https://youtu.be/0r9o8hprDXQ)

This video said about how to choose number of filters, filter size effects (the first min). So, you will need to increase the size of the filter when you wan to increase the size of detected patterns.

What is "detected patterns"?

So, this is intuition behind what happens when CNN trained: Given a CNN has 3 CONV layers. The first layer is for discovering simple shapes like lines, blobs. The second layers recognizes basic geometric shapes. The last CONV layer pick out the complex combination features from the previous layer that are most valuable features should be used for classification. What CNN discovers in each layer is detected patterns.

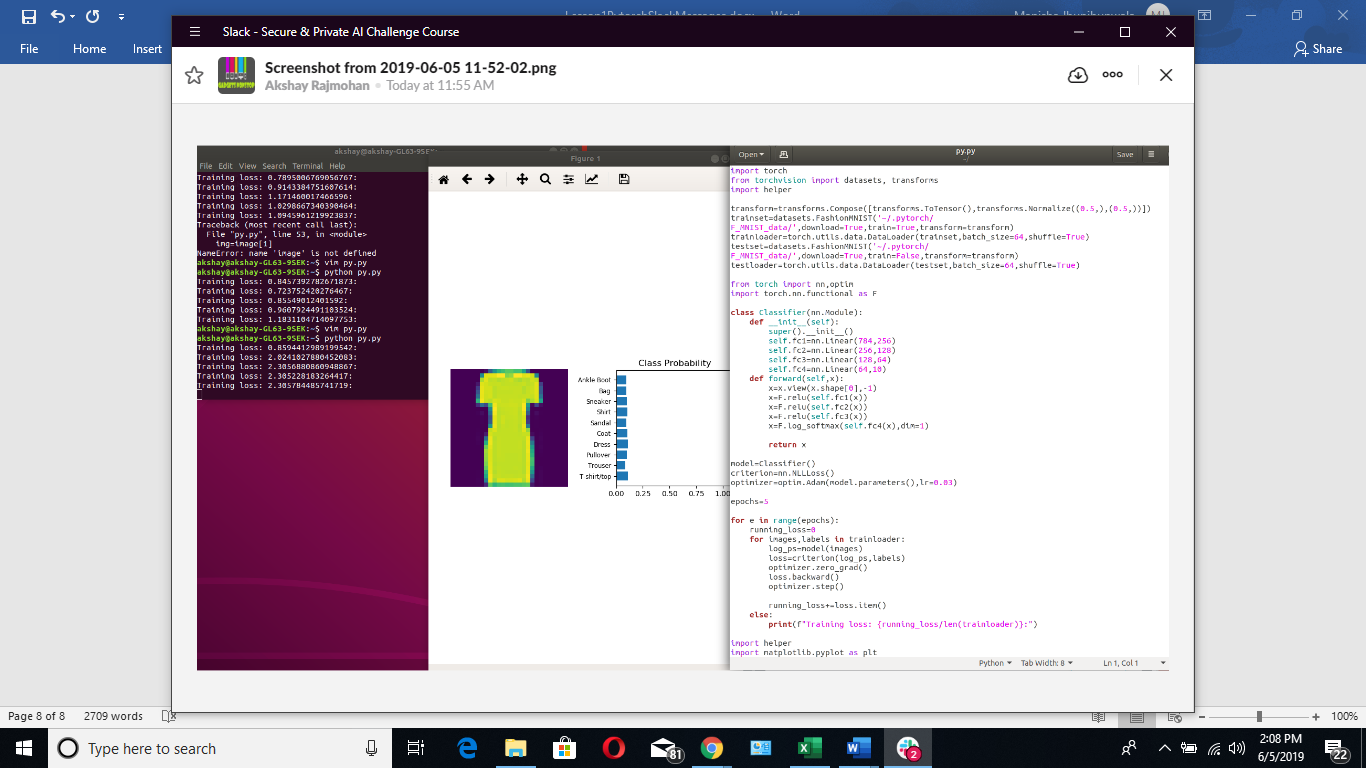
## **how to make pytorch use my gpu?since i am using laptop i have 2 gpus,one is intel and other one is nvidia RTX**

~1. Install Nvidia's drivers

2. `sudo prime-select nvidia`

[**https://towardsdatascience.com/speed-up-your-algorithms-part-1-pytorch-56d8a4ae7051**](https://towardsdatascience.com/speed-up-your-algorithms-part-1-pytorch-56d8a4ae7051) **- Speed Up your PyTorch models**

## **is this correct?cause it seems like my training loss decreases then increases and the predictions are not so good it seems. Is there any problem in the code?**



Code seem to be fine.. try decreasing lr and increasing epochs.. this model cannot find the global minima and is oscillating about the local minima try to fix it by decreasing lr

you need to use optimizer.zero\_grad() before start of minibatch i.e. before calculating log\_ps

To my knowledge, optimizer.zero\_grad() resets gradients, not the weights, so has no effect in calculating the prediction, only backprop. If I am correct, it is needed only before loss.backward() and optimizer.step(). Am I right guys?

## **why do we print loss as runningloss/len(trainloader)?**

Remeber the total running\_loss is being updated for each batch.

so that we can make it scalable .otherwise the loss will depend on the amount of the input data. consider your batch size is 32 and 64 if you don't divide it by the len(trainloader) you will get different loss value.so we divide it so that it does not depend on how much data we are giving in as input

We are calculating running loss for a unit (defined as an epoch) by dividing runningloss and len(trainloader).

To compute loss for each epoch and then use it for updating the parameters ( caution: parameters are weights, bias,etc. This are different from hyperparameters)

To track how the model is performing after a given iteration. A better way to visualize is to plot graph between loss and no of iteration, if you loss is decreasing over time. Then your model is performing quite well.

Simply, we print the loss to check whether our training is heading in the correct direction such as (We must expect our training loss is decreasing rather than increasing). If the training loss is increasing or stable, there is something wrong and we could just stop it immediately rather than waiting the whole training has finished. It saves time and if you are using GPU, it saves your money too.

the formula indicates loss per batch of images encountered in that epoch/event.

## **can anyone explain this to me ? Code of initializing the weights, biases and features of NN.**

**### Generate some data**

**torch.manual\_seed(7) # Set the random seed so things are predictable**

**# Features are 3 random normal variables**

**features = torch.randn((1, 3))**

**# Define the size of each layer in our network**

**n\_input = features.shape[1] # Number of input units, must match number of input features**

**n\_hidden = 2 # Number of hidden units**

**n\_output = 1 # Number of output units**

**# Weights for inputs to hidden layer**

**W1 = torch.randn(n\_input, n\_hidden)**

**# Weights for hidden layer to output layer**

**W2 = torch.randn(n\_hidden, n\_output)**

**# and bias terms for hidden and output layers**

**B1 = torch.randn((1, n\_hidden))**

**B2 = torch.randn((1, n\_output))**

torch.manual\_seed (7) : this cmd helps us to keep the random variables same each time when you call the function again.

Features statement: generates tensor with random values with 1 row and 3 colums ( tensor are 3d matrix)

N\_input: takes the shape same as that of features tensor (i.e.(1,3))

N\_hidden: 2 neuron in hidden layer

N\_o/p: 1 neuron in o/p layer (binary classification)

W1- defining random weights of shape (n\_input,n\_hidden)

W2- same as W1

B1- defining random bias of shape (1,n\_hidden)

B2- SAME AS B1

## **I read `The only requirement is that for a network to approximate a non-linear function, the activation functions must be non-linear.` in the second notebook and couldn't really understand it. Can anyone explain?**

Activation functions and it’s types-Which is better?” by Anish Singh Walia

<https://link.medium.com/51jdkzpugX>

Normally, when we plot our training data, we can see that the boundary separating them is non-linear i.e. not a straight line. Hence, to make sure that the network approximates a non-linear function to predict the output, we use non-linear activations. This makes sure that the neural networks learn the appropriate function to model the output.

<https://www.coursera.org/lecture/neural-networks-deep-learning/why-do-you-need-non-linear-activation-functions-OASKH>

Real-world data is mostly non-linear, so using a non-linear activation function helps to mimic the non-linearity in that data. All that is happening behind the scenes is a series of mathematical operations, and using a simple linear function does not capture the weirdness and complexities of real world data.

Take the sigmoid activation function used for classification problems. It helps to determine if a particular class is True/False. If we use a Linear, we just deal with linear values, and that does not help with classification.

You would more normally use Linear functions for problems of \*regression\*, and \*non-linear\* for other problems like \*classification problems\* (edited)

Activation functions cannot be linear because neural networks with a linear activation function are effective only one layer deep, regardless of how complex their architecture is.

## **Hey, what's the difference in using activation functions from nn.functional or nn? For, example, using F.relu or using nn.ReLU I believe this question goes a bit in hand with using nn.Sequential or creating a class for the network**

<https://discuss.pytorch.org/t/what-is-the-difference-between-torch-nn-and-torch-nn-functional/33597/2>

While the former defines nn.Module classes, the latter uses a functional (stateless) approach. To dig a bit deeper: nn.Modules are defined as Python classes and have attributes, e.g. a nn.Conv2d module will have some internal attributes like self.weight. F.conv2d however just defines the operation and needs all arguments to be passed (including the weights and bias). Internally the modules will usually call their functional counterpart in the forward method somewhere.

F.relu is the functional interface of torch.nn.ReLU. Modules like torch.nn.ReLU are sometimes handy, for example when quickly creating a model using nn.Sequential.

You can’t add F.relu in a nn.Sequential, as it expects an object that is inherited from nn.Module.

nn.ReLU is expected in nn.sequential because we can't use F.relu there. But it is easier to perform complex task with F.relu

Similarly for complex task and more functionality class is created but if you want an straight forward sequential operation implementation we use nn.sequential

In addition to above answer of @Stark. We can suppose that there are two methods of defining the activation functions F.Relu is one method which is used in the functional interface of torch.nn.Relu. this modules do not requires all attributes to passed to the function whereas in 2nd case, nn.sequential we require to pass all the attributes.

Personally, I think creating activation, dropout, pooling etc. modules in \_\_init\_\_ makes it easier to reuse the model. For example, when extracting features, you may wish to wrap a pretrained model and overwrite the forward function to return the feature variables required. Having these modules let you conveniently do this instead of inserting many functional calls.

But using functional interfaces we may do some fancy operations like convolving two feature maps explicitly with F.conv2d.

## **Hi! one question guys.. why do we need to transform and normalize data set??**

Normalization helps to work with values within a particlar range. Sometimes working with very large number will result in Overflow or Runtime errors (

<https://medium.com/@urvashilluniya/why-data-normalization-is-necessary-for-machine-learning-models-681b65a05029>

Consider how a neural network learns its weights. C(NN)s learn by continually adding gradient error vectors (multiplied by a learning rate) computed from backpropagation to various weight matrices throughout the network as training examples are passed through.

The thing to notice here is the "multiplied by a learning rate".

If we didn't scale our input training vectors, the ranges of our distributions of feature values would likely be different for each feature, and thus the learning rate would cause corrections in each dimension that would differ (proportionally speaking) from one another. We might be over compensating a correction in one weight dimension while undercompensating in another.

This is non-ideal as we might find ourselves in a oscillating (unable to center onto a better maxima in cost(weights) space) state or in a slow moving (traveling too slow to get to a better maxima) state.

It is of course possible to have a per-weight learning rate, but it's yet more hyperparameters to introduce into an already complicated network that we'd also have to optimize to find. Generally learning rates are scalars.

Thus we try to normalize images before using them as input into NN (or any gradient based) algorithm

## **Hi,can anyone explain me what does these below 2 lines does %matplotlib inline %config InlineBackend.figure\_format = 'retina'**

I believe `%matplotlib inline` is used to ensure the the plots are shown as part of the jupyter notebook

%config InlineBackend.figure\_format = 'retina' - gives you retina-display quality for figures into code

**%matplotlib inline is used to render figures directly in the notebook, you should use the inline backend with the command. On higher resolution screens such as Retina displays, the default images in notebooks can look blurry. Use %config InlineBackend.figure\_format = 'retina' after %matplotlib inline to render higher resolution images.**

## **is there any other option than Retina display**

**For SVG images use %config InlineBackend.figure\_formats=['svg']**

[**https://ipython.org/ipython-doc/2/config/options/notebook.html**](https://ipython.org/ipython-doc/2/config/options/notebook.html)

From the Docs

InlineBackend.figure\_formats : Set

Default: set([‘png’])

A set of figure formats to enable: ‘png’, ‘retina’, ‘jpeg’, ‘svg’, ‘pdf’.

This turns on inline plotting, where plot graphics will appear in your notebook. This has important implications for interactivity. For inline plotting, commands in cells below the cell that outputs a plot will not affect the plot. For example, changing the color map is not possible from cells below the cell that creates a plot. However, for other backends, such as Qt5, that open a separate window, cells below those that create the plot will change the plot - it is a live object in memory.

## **Is it a good practice to create large NN for small datasets ?**

Not at all NN itself is an overkill for small datasets, large NN are not at all preffered, when we say small there are two aspects :

1.) dimensionality of the data

2.) literal size of the data

if it is 1.) NN is fine but you will have to do a lot of hyperparameter tuning or bring down the dimensionality drastically and use trees

If it is only 2.) and not 1.) trees outsmart NN in fact randomForest outsmarts all the algorithms for tabular data, where dimensionality is low (there is a 2016 research on this).

As others already mentioned, NN tends to overfit small datasets. You should go with Decision Trees and Random Forest. They work wonders if implemented correctly. Fun fact: XBox Kinect uses Random Forests to predict the approximate 3-D location of a user based on the sesnsor data.

<https://beamandrew.github.io/deeplearning/2017/06/04/deep_learning_works.html>

Recently I did a project, where the dataset wasn't considerably large. The implementation of the exact same code in the below link helped to achieved very high accuracy inspite. The implementation is Lasso Regression, where we can cross-validate it using AIC and BIC criterion and choose the best out of it. The following implementation is using scikit-learn. I believe the same can be done using PyTorch also.

<https://scikit-learn.org/stable/auto_examples/linear_model/plot_lasso_model_selection.html>

There's a theorem called Occam's razor, which states choosing the simplest hypothesis that can solve your problem. You should always go for the simplest learning algorithm that solves your problem.

## **Can anyone tell me, whats the advantage of using object oriented model over Sequential model ? one think I know is OO model can be used for more complex model such as resnet etc. , But has there any advantage or disadvantage in case of performance such as seed etc..**

Pytorch requires OO model bcause it makes use of those constructs which you write like forward and backward and getitem in its core modules. You cannot go about creating a simple sequential program with no standard regular programming constructs if you want to use a framework. The OO model ensures everyone uses the same constructs so that it can automate a huge burden (automatic update of gradients), which otherwise consumes lot of development time

Object-oriented model is a representation of a piece of software as a set of objects interacting between each other, with a goal to reduce the complexity of the system and enable developers to work on a specific object, while treating other objects as black boxes, with the requirement to know only their interfaces, and not their actual implementation.

Some time ago, I had the same question and was considering whether there was a difference in speed among the two. So, I went on the PyTorch forums and found this.

<https://discuss.pytorch.org/t/differences-between-writing-models-with-nn-sequential-vs-creating-a-class/17470>

It turns out there is no difference. OOP model has more boilerplate code but it gives more control over the flow of the data in the network. But Sequential is for simpler models and faster to construct for the programmer. I didn't find any difference in speed and efficiency among the 2 anywhere at that time.

Programming languages fall into two categories: sequential and concurrent. Sequential languages are languages that were designed for writing sequential programs and have no linguistic constructs for describing concurrent computations. Concurrent programming languages are languages that were designed for writing concurrent programs and have special constructs for expressing concurrency in the language itself.

While OOP is a paradigm. OOP brings the concept of class which is a collection of data and methods that relies on the operation of object.The concept of class and object brings the dynamicness within a code and most importantly make the code reusable unlike procedural languages. Features of it include: Inheritance,Encapsulation,Polymorphism, Abstract classes etc.

If you are asking of nn.Sequential then the major problem is you can't later extract the weights easily and also sequential uses nn module functions rather than the nn.functional module functions in activation and regularization layers these offer more better control for training than doing it sequential.

Try to think how you would implement a skip connection with nn.Sequential (see <https://en.wikipedia.org/wiki/Residual_neural_network>).

Inheriting nn.Module allows you to do this easily.

## **Can someone provide me some examples of parameters and hyperparameter? I am trying to figure out the difference between these two. Thanks**

parameters are variable values such as weights and biases associated with the nodes in a neural network (

parameters are the values the model adjust in the learning process and hyperparameters are the values set before the learning process begins, examples parameter: weights, hyperparameter: amount of hidden layers

**Model parameters are the properties of the training data that are learnt during training by the classifier or other ml model. For example in case of some NLP task: word frequency, sentence length, noun or verb distribution per sentence, the number of specific character n-grams per word, lexical diversity, etc. Model parameters differ for each experiment and depend on the type of data and task at hand.**

**Model hyperparameters, on the other hand, are common for similar models and cannot be learnt during training but are set beforehand. A typical set of hyperparameters for NN include the number and size of the hidden layers, weight initialization scheme, learning rate and its decay, dropout and gradient clipping threshold, etc**

imply puts -> if you change the value and it affect all other parameters then it is hyper-parameters (learning rate, no. of hidden units, no. of layers, model architecture and so on). During training, you need to tune those parameters. While parameters are computed by the algorithms such as weights and biases.

## **weights.reshape(a, b) will return a new tensor with the same data as weights with size (a, b) sometimes, and sometimes a clone, as in it copies the data to another part of memory". Can anybody please explain more on \*sometimes\* part of the statement?**

reshape has two options

1. inplace - it won't give u new tensor

2. outofplace - it copes data into new tensor

torch.reshape() returns a view if the shapes are compatible, and copies (equivalent to calling contiguous()) if the shapes are not compatible.

Here, Sometimes refers to if same number of elements then it returns inplace memory otherwise outplace memory.

\*weights.reshape(a, b)\* will return a new tensor with the same data as weights with size (a, b) as in it copies the data to another part of memory.

\*weights.resize\_(a, b)\* returns the same tensor with a different shape. However, if the new shape results in fewer elements than the original tensor, some elements will be removed from the tensor (but not from memory). If the new shape results in more elements than the original tensor, new elements will be uninitialized in memory.

Sometimes" is when data is not contiguous (continuous block in memory) and neither in compatible strides (data stored in memory that cannot be accessed in any pattern ), then it will copy data into some other part of memory and return reshaped tensor. (This is my perception)

## **Hi guys ! I am in the Part 15: Classifying Fashion-MNIST and I made the exercise. However, I have some questions about the output layer activation function and the criterion for the loss function. I understand that softmax and logsoftmax activation functions are useful when one is computing probabilities for classification. What I don't understand is how to choice a good loss function for that kind of problems and what benefits those functions give us in the solution of a classification problem. Thanks a lot !**

You can take a look here:

<https://medium.com/udacity-pytorch-challengers/a-brief-overview-of-loss-functions-in-pytorch-c0ddb78068f7>

In many cases, \_Mean Squared Error Loss\_ for regression problems and \_Cross Entropy Loss\_ (or \_Binary Cross Entropy Loss\_) for classification problems work well.

I do see many sophisticated research papers using custom loss functions that involve several regularization terms and varying coefficients, but when you are having difficulties with the training—unless it's due to using a loss function that doesn't fit at all to the task or output activation—it's often better to check the data, use a different weight initialization, or try different hyperparameters such as architectures of the neural network models or learning rates.

## **Can someone explain when to use normalisation and standardisation? I know the difference between these 2 but am not sure when to use normalisation and when to use standardisation. Tried googling but to no avail too.**

Normalization typically means rescales the values into a range of [0,1]. Standardization typically means rescales data to have a mean of 0 and a standard deviation of 1 (unit variance).

There is a good article on this here:

<https://towardsdatascience.com/normalization-vs-standardization-quantitative-analysis-a91e8a79cebf>

So the way I see it and to keep it very simple: Imagine your dataset has 2 features, X1 and X2. Now X1 range is from 0 to 1million while X2 range is from -1 to 1. Maybe both features are related, but when you do certain math operations between them, the orders of magnitude are very different, and that might lead to one feature diminishing the other. So one way to cope with that is to normalize or standarize the features in the same order of magnitude.

This might have many implications, faster convergions, more accuracy, less accuracy, etc. Depending on the nature of your data is whatever is suitable or not!

Also, a huge issue with normalization is that it's very sensitive to outliers.

If your dataset contains outlier(s) with abnormally low or high values for the features, normalization will put the non-outliers into almost the same value. (For example, if there's a data point with value `9999` while all the others are around `1`, then the value of the outlier would be `1` and all the others' values would be practically `0`s.)

Of course, if there are only few outliers and you are sure that those data points are garbage, you can manually take them out, but, HEY, we are trying to get the machines do as much work as possible in stead of humans!)

(Even in standardization, outliers would have negative effects. However, standardization is less sensitive to these data since they also considers the variance of the data—or how the data points are distributed.)

Normalization is good to use in machine learning and deep learning in order to converge more rapidly and precisely since you might have many attributes in your data with different minimum and maximum values, some of them too large.

standardize in multivariate analysis when you want all variables to be in comparable units.

Normalize can mean to fit a normal distribution to a set of data or also to transform a variable to a normal.

<https://medium.com/@rrfd/standardize-or-normalize-examples-in-python-e3f174b65dfc>

## **Hi everybody! I'm in Part 7 and I was wondering if for data augmentation, besides all the transformations mentioned, would it be helpful to add images with some blur? could it help to make the Neural Network more robust?**

It’s worth to do it if you want your model robust enough to recognize such images.

You can try to add some Gaussian blur as augmentation. There are also other non-conventional techniques.

Neural Network tries to learn data features. You provided the dataset with noisy images and train your neural network with this data, so, your network learns these noisy features which will effect in the classification process with the testing dataset.

1. **I am having a hard time understanding Inference and validation**.

you have a trained network, you can use it for making predictions. This is typically called inference, a term borrowed from statistics. However, neural networks have a tendency to perform too well on the training data and aren't able to generalize to data that hasn't been seen before.This is called overfitting and it impairs inference performance. To test for overfitting while training, we measure the performance on data not in the training set called the validation set. We avoid overfitting through regularization such as dropout while monitoring the validation performance during training.

Figuratively,

\*Validation\*: Quiz

\*Test\*: Final Exam

\*Inference\*: Real world problem solving using what you learned

---------------

You use a \*validation\* set to see how your model is learning during the training (mainly to see whether your model is generalizing well or simply overfitting to the training data).

(Usually, after the training, you'd give the model another separate set of data as a test set. This is helpful in case your model isn't generalizing but gets lucky with the validation set.)

I can't really find a concrete definition for \*inference\*, but as far as I can see, it usually refers to making useful predictions with \*trained\* model(s).

Validation is done during training time to see the performance of the model on data that is not used for training (but this is done during training). For example, Consider your dataset as "Dataset = a+b+c+d". Here a, b, c and d are the broken set of the training dataset. Now for training you will use a+b+c and for validation you will use 'd'. This is also called as Cross-Validation. Similarly, a different combination of a/b/c/d can be used for trainig+Validation. While the Inference is making prediction on completely unseen dataset (test data).

## **Hi all, I'm wondering about the image resizes w/ pretrained models: why the number 224 for size? Why not 256 or other?**

According to the Alexnet paper, the choice of 224x224 was due to their use of data augmentation techniques (translations and reflections) on the 256x256 images

Ok, so basically 224 is the sweetspot because not much information is sacrificed, yet we are able to perform data augmentation on them. Is it?

all the pretrained models we use are trained on imagenet dataset where images are of 224\*224 with mean and std of

[0.485, 0.456, 0.406]. So when we are reusing it we should use same things to get better results

224\*224 is the input image format for the alexnet

## **Hi everybody, I have a question, how is it different to use transpose, instead of view, reshape and resize?**

Reshape gives you new tensor, with the same data but a different dimension, while resize does an in-place operation and returns the same tensor with a different shape. Then View returns new tensor with specified size (a,b). But transpose will just exchange or alter the dimension by row to col/ col to row

Imagine your matrix is a square jpg image:

Transpose: rotate 90 degrees clockwise, and flip \_horizontally\_ to the left side.

View, reshape, resize: just change the width and/or height

It's throwing an error because you didn't pass the shapes in a tuple

## **How to choose the number of hidden layers? Are there any rules?**

check out this :

<https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw>

The number of hidden units should be more experimental but depending on how complex the data and the problem is, it should be deep.

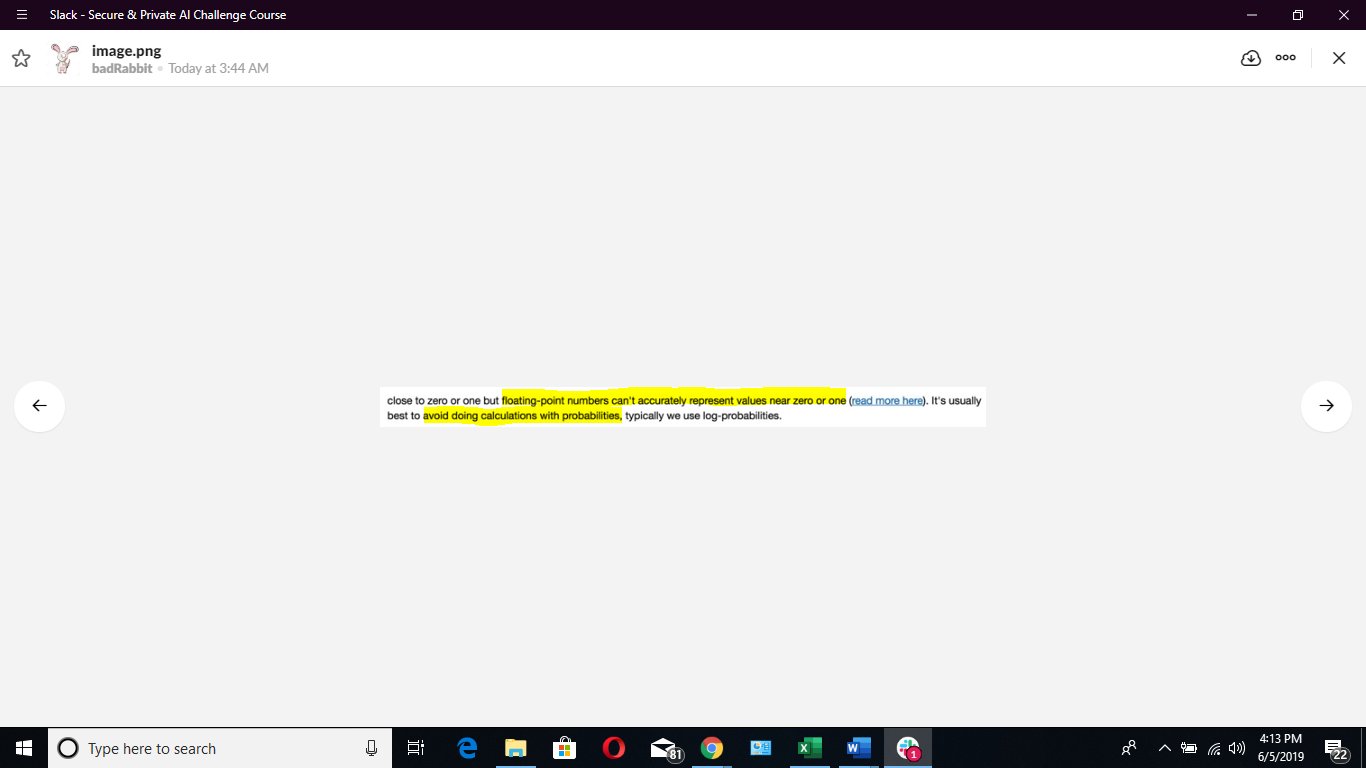
It is usually the power of two btw. Like 32, 64, 128, and so on.

i guess you're talking about hidden units, is it the same for the number of layers ?

<https://www.google.ca/amp/s/www.researchgate.net/post/How_to_decide_the_number_of_hidden_layers_and_nodes_in_a_hidden_layer/amp>

hidden layers and hidden units are different things. There is not a unique answer to the correct number of hidden layers. It depends of many variables. If you are working with images you will need more layers. But anything else the more number of layers more chance of overfitting. So every case is a case.

## **Explain The highlighted part**



From what I undestand, basically every CPU or GPU has a floating point precision limit, beyond that limit it can't represent smaller float numbers or very big numbers. When doing aritmethics with floats, the problem might happen if you are computing with two numbers which result is very close to those limits, or after some decimal position the result is not precise.

This is related with the binary representation on memory of floating point numbers and single and double precision This link might give some more insights:

<http://cstl-csm.semo.edu/xzhang/Class%20Folder/CS280/Workbook_HTML/FLOATING_tut.htm>

The computer uses a limited digit floating point representation of fractions, multiplying so many probabilities is guaranteed to be very very close to zero.

With log, we don't have this issue.

## **transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)), ]) can anyone explain this?**

Hi! Take a look at this thread! Might help!

<https://secureprivataischolar.slack.com/archives/CJCJ5UVH8/p1559599836345500>

`transforms` is used for apply transformation to your data. `transforms.Compose` crates a pipeline that apply a series of transformation in a row.

`transforms.ToTensor()` transform your input data into a tensor. Then `transforms.Normalize` normalize the tensor by subtract it by a value (the first parameter, usually its average) then divide it by another value (the second parameter, usually its standard diviation). In this case there are 3 value like [0.5, 0.5, 0.5] because there are 3 color channels in a picture. RGB.

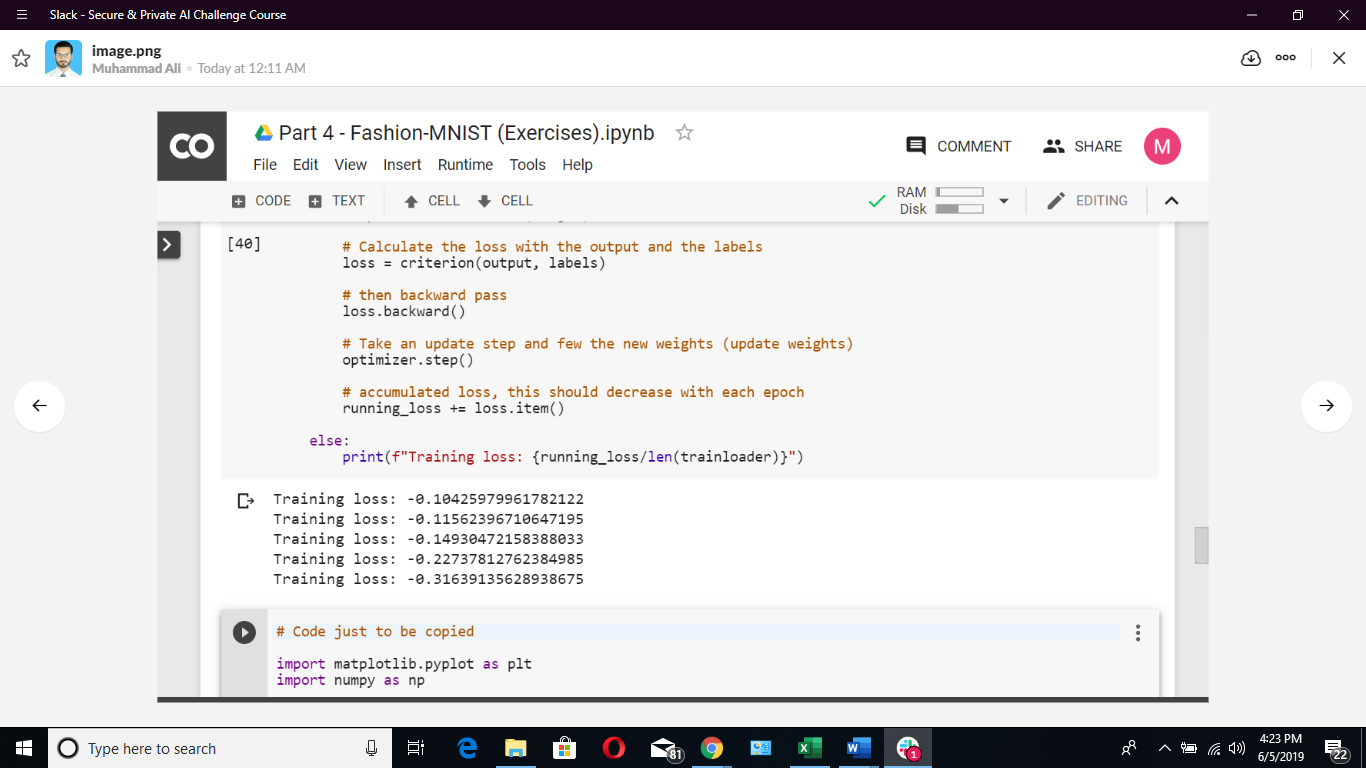
Normalize does the following for each channel:

image = (image - mean) / std

The parameters mean, std are passed as 0.5, 0.5 in your case. This will normalize the image in the range [-1,1]. For example, the minimum value 0 will be converted to (0-0.5)/0.5=-1, the maximum value of 1 will be converted to (1-0.5)/0.5=1.

## **I am getting -ve loss, Help please! do mention my mistake as it will allow me to learn and correct it, Thanks in advance :smile: @Arka @Taimur Zahid**

P.S. the -ve loss is increasing that means that overall loss is decreasing which is kinda good thing :



Is your output activation function a log\_softmax?

Maybe @Muhammad Ali you are using logsoftmax in the output and crossentropyloss in the loss function. That can be the issue

Can you kindly guide me why it caused this unusual behaviour? and why NLLLoss and Softmax can't work together?

As per the documentation, NLLLoss expects the input in the form of logarithmic probabilities, that's why they have mentioned that for NLLLoss(), the last layer of your network should be LogSoftmax(). If you want to use softmax() for your final layer, then you can use CrossEntropyLoss().

I don't really know it's implementation but one thing I know is the crossentropyloss is a combination of the log\_softmax and NLLLoss. So when you use crossentropyloss on softmax, it first converts to log\_softmax then applies NLLLoss

## **why labels in MNIST is showing 64, rather if there are ten classes then it should be showing 10 labels and images.shape be showing 10,1,28,28 ?**

Nope it should be [batch\_size,1,28,28]

If you print labels.shape you will find [batch\_size,10] as output

what does the 1 represent?

1 represents no. Of channel. As it is a grey scale image the channel is 1. If you use colour image you will find it as 3

## **What does mean, that calculates the loss with logits and he labels,? What is logits. It's confusing.**

Logits are the output to which the softmax are not applied yet. They are the unnormalized (not activated) output of a model.

the raw output (output before passing it ti softmax) is called logits. We use the logits because softmax gives you probabilities which will often be very close to zero or one but floating-point numbers can't accurately represent values near zero or one

It is, not zero-like values, so it helps in computation better

Logit is a function that maps probabilities ( [0, 1] ) to R ( (-inf, inf) ) Probability of 0.5 corresponds to a logit of 0. Negative logit correspond to probabilities less than 0.5, positive to > 0.5.

## **Why we set attribute requires\_grad= True in tensor cmd it is value is set to true because autograd works by keeping track of operations performed on tensors, then going backwards through those operations, calculating gradients along the way. so keep track on tensor we use this attribute. \*this is what I understood regarding this attribute any correction is welcomed\***

We don't need gradient calculations and parameter updates during testing phase, so we simply disable the weight optimizer functionality for that duration

Requires\_grad=True allows the gradient flow across the network. So gradient of loss in respect to every Weight will be calculated. Then optimizer will tune those weight

## **Does anyone has any idea what is linear perceptron?**

A network without any activation functions. The output at each layer is just a result of basic math operations like addition and multiplication.

look a percepton is a Linear classifier and Logitic Regression is also a Linear classifier both are represented mathematically similarly

i don't know how to write those methods representation due to complex symbols

## **What is super().\_\_init\_\_(self) ??**

**In :-**

**class Network(nn.Module):**

**def \_\_init\_\_(self):**

**super().\_\_init\_\_()**

**self.fc1 = nn.Linear(inSize, 256)**

**self.fc2 = nn.Linear(256, 128)**

**self.fc3 = nn.Linear(128, 64)**

**self.fc4 = nn.Linear(64, 10)**

**def forward(self, x):**

**x = F.relu(self.fc1(x))**

**x = F.relu(self.fc2(x))**

**x = F.relu(self.fc3(x))**

**x = F.log\_softmax(self.fc4(x), dim=1)**

**return x**

It is initializing the Parent class which is nn.Module.

super is calling its parent call through inheritance

You’re basically making your Network class to be a PyTorch module. All Network classes should inherit from the nn.Module parent class.

Just like intializing parent class constructor in java or cpp.

Here parent class' initializer method is called and all its variable and methods in self \_\_init\_\_ () are initialized .

Here are two links which explain super().\_\_init\_\_(self)

<https://realpython.com/python-super/>

<https://www.pythonforbeginners.com/super/working-python-super-function>

## **what is x in forward method ?**

**def forward(self, x):**

**# Pass the input tensor through each of our operations**

**x = self.hidden(x)**

**x = self.sigmoid(x)**

**x = self.output(x)**

**x = self.softmax(x)**

X is the inputs to the network .

X is the input feature tensor

x at the begining is the input but you are reassigning it through the layers

## **What is self?**

Self represents the instance of the class. By using the "self" keyword we can access the attributes and methods of the class. Here x is an input but you are getting updates from each layer, thus you are re-assigning it. This instance caries all the updates when a function is called with an input.

## **Why we iterate the trainloader? And what exactly the trainloader is? Clarification will be helpful!?**

trainloader is like a linked list. Each position will point to values (Images) equivalent to the batch size. You iterate over it to access each batch. We iterate it because we cannot access each position through subscript like an array. You have to use the next() to move to the next position. Hope it makes sense.

The trainloader is an instance of the torch.utils.data.DataLoader() class. The DataLoader class combines the dataset with a sampler, so that rather than working on the whole dataset, we can work on batches extracted from the dataset. We access the DataLoader object by iterating through it.

You can consider Trainloader to be a list of batches of images. We pass each batch (32/64 images typically) through forward pass, loss, back propagation steps. Thus, we iterate through all the batches (which is called one epoch).